AMENDMENT TO THE SPECIFICATION

Please replace the paragraph beginning at page 11, line 1 and ending at page 11, line 14 with the following paragraph:

-- particular, the current implementation employs an MDES-driven compiler that can be re-targeted to a specific machine being designed based on the MDES constructed by the composite MDES extractor 24. The compiler input includes an MDES representation of the opcode repertoire and the resources that are used for each operation. Using this input, the compiler is re-targeted for the machine. It can then be used to schedule an application program and generate statistics about the use of the opcodes that are useful in customizing the machine's design. This is particularly useful in the design of application specific processors that are designed for a particular type of application program. For more information about the MDES extractor, please see co-pending U.S. Patent Application No. 09/378,601, filed August 20, 1999, now U.S. Patent No. 6,629,312, by Shail Aditya Gupta entitled PROGRAMMATIC SYNTHESIS OF A MACHINE DESCRIPTION FOR RETARGETING A COMPILER, which is hereby incorporated by reference.--

Please replace the paragraph beginning at page 11, line 15 and ending at page 11, line 24 with the following paragraph:

-- The instruction format designer 22 generates instruction formats based on the datapath design and the abstract ISA spec. In particular, it builds a syntax tree called the instruction format (IF) graph 26, and then uses the tree to allocate bits in the processor's instruction unit to the fields of VLIW instructions. For more information about this process, please see co-pending U.S. Patent Application No. 09/378,293, filed August 20, 1999, now

U.S. Patent No. 6,457,173, by Shail Aditya Gupta, Bantwal Ramakrishna Rau, Vinod Kumar Kathail, Richard Craig Johnson and Michael S. Schlansker, entitled AUTOMATIC DESIGN OF VLIW INSTRUCTION FORMATS, which is hereby incorporated by reference. --

Please replace the paragraph beginning at page 12, line 1 and ending at page 12, line 7 with the following paragraph:

-- process, the controlpath design fills in the control components and signals to the datapath design already in the AIR 32 representation of the processor. For more information on the control path design process, please see co-pending patent application no. 09/378,394, filed August 20, 1999, now U.S. Patent No. 6,490,716 by Shail Aditya Gupta and Bantwal Ramakrishna Rau, entitled, AUTOMATED DESIGN OF PROCESSOR INSTRUCTION UNITS, which is hereby incorporated by reference.--

Please replace the paragraph beginning at page 13, line 17 and ending at page 13, line 34 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

-- As an example, a simple 2-issue machine is specified below. This example specification is expressed in a database language called HMDES Version 2. See John C. Gyllenhaal, Wen-mei W. Hwu, and B. Ramakrishna Rau. HMDES version 2.0 specification. Technical Report IMPACT-96-3, University of Illinois at Urbana-Champaign, 1 996. This language organizes the information into a set of interrelated tables called sections containing rows of records called entries. Each entry contains zero or more columns of property values called fields.

```
SECTION Operation Group {
   OG_alu_0 (ops (ADD SUB) format (OF_intarith2));
   OG_alu_1 (ops (ADD SUB) format (OF_intarith2));

   OG_move_0 (ops (MOVE) format (OF_intarith1));
   OG_move_1 (ops (MOVE) format(OF-intarith1));

   OG_mult_0 (ops (Mpy) format (OF_intarith2));
   OG_shift_1 (ops (SHL SHR) format (OF-intarithi));
  } --
```

C4

Please replace the paragraph beginning at page 14, line 1 and ending at page 14, line 4 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

```
-- SECTION Exclusion_Group {
   EG_0 (opgroups (OG_alu_0 OG_move_0 OG_mult_0));
   EG-1 (opgroups (OG_alu_1 OG_move_1 OG_shift_1));
   }--
```

C5

Please replace the paragraph beginning at page 14, line 5, and ending at page 14, line 21 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

-- This example specifies two ALU operation groups (OG_alu_0, OG_alu_1), two move operation groups (OG_move_0, OG_move_1), one multiply group (OG_mult_0), and one shift group (OG_shift_l). These operation groups are further classified into two exclusion groups (EG_0, EG_1) consistent with a two-issue machine. The multiply group shares resources with one ALU group, while the shift group shares resources with the other. Each operation group also specifies one or more operation formats shared by all the opeodes within the group. Additional operation properties such as latency and resource usage may also be specified, as shown below.

```
SECTION Operation_Group (
        OG_alu_0        (ops (ADD SUB) format("OF-intarith2")
                        latency (OL_int
                        resv (RT_OG_alu_1)
                        alt_priority (O));
}--
```

Please replace the paragraph beginning at page 15, line 17 and ending at page 15, line 35 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

-- The code listing below provides an example of the register file and operation format inputs sections of an ArchSpec:

```
SECTION Register_File (
  gpr(width(32) regs (r0 rl ... r3l) virtual(I));
  pr(width(l) regs (p0 p1 ... p15) virtual(P));
```

lit(width(16) intrange (-32768 32767) virtual(L));

SECTION Field_Type

FT_I(regfile(gpr));

FT_P)regfile(pr));

FT_L(regfile(lit));

FT_IL(compatible-with(FT-I FT-L));

C7 }

SECTION Operation_Format

OF -intarith1 (pred (FT P) src (FT_I) dest (FT 1));

OF -intarith2 (pred (FT-P) src (FT_IL FT_I) dest (FT-I));

}--

Please replace the paragraph beginning at page 16, line 12 and ending at page 16, line 19 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

-- The specification may also contain information defining additional architecture parameters:

C8

SECTION Architecture_Flag {

predication_hw (intvalue (l));

speculation_hw (intvalue (0));

systolic_hw (intvalue (l));

technology_scale (doublevalue (0.35));

}--

Please replace the paragraph beginning at page 22, line 5 and ending at page 22, line 29 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

-- After building the exclusion matrix, the datapath synthesizer executes a recursive algorithm on the matrix data to find the exclusion cliques. The exclusion graph naturally follows from the exclusion relationship expressed in the matrix. The recursive algorithm operates on this graph according to the following pseudocode:

```
RecursiveFindCliques(currentClique, candidatenodes)
1:  // Check if any candidate remains
2:  if (candidatenodes is empty) then
3:  // (check it the current set of clique nodes is maximal
4:  if (currentclique is maximal) then
5:  Record(currentClique);
6:  endif
7: else
8:   StartNodes = Copy(candidateNodes);
9:  while (StartNodes is not empty) do
10: Hl: if (currentclique ∪ candidatenodes ⊆ some previous Clique) break
11:     node = pop (StartNodes);
12:     candidatenodes = candidatenodes - [nodes];
13:     if (currentclique ∪ (node) is not complete) continue;
14:     H2: prunedNodes = candidateNodes ∩ NeighborsOf (node);
```

15: RecursiveFindCliques (currentClique ∪ {node}, prunednodes);

16: H3: if (candidatenodes) ⊆; NeighborsOf (node) break;

17: H4: if (this is first iteration) startNodes = startNodes-neighborsOf (node);

16: endwhile

17: endif --

Please replace the paragraph beginning at page 24, line 3 and ending at page 24, line 34 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

-- After finding maximal sets of mutually exclusive operation groups, the datapath synthesizer selects functional units from a standard or user-specified macrocell library so that all of the opcodes occurring in each set are covered, i.e., able to be executed on the selected functional units. As shown in FIG. 3, the current implementation selects functional units to cover the exclusion cliques (see step 42). Next, the datapath synthesizer instantiates the selected functional units as shown (step 46). In building the functional units in this manner, the objective is to optimize the selection of functional unit instances so that all of the required opcodes are still supported while maintaining the exclusion requirements defined by the cliques. In some cases, it may not be possible to map individual cliques to a single functional unit, thereby necessitating the use of multiple functional units to support the opcode requirements of the clique. Pseudocode for covering the cliques and building the functional units is listed below:

BuildFUs (VLIWArch, listOfCliques)

1: foreach (OPG ∈ VLIWArch)

2: build valid ListOfFUs(Opset(OPG)) from Database;

3: // match opcodes, latency

```
4:    foreach (OPG ∈ VLIWArch)
5:    foreach (usedFU ∈ ListOfFUs(Opset(OPG)))
6:     ListOfOpsets(usedFU) += Opset(OPG);
7:    while (listOfCliques is not empty)
8:    find (bestFU ∈ usedFUs) such that
9:    forsome (clique ∈ listOfCliques)
10: maxCoveredOPGs = {OPG | OPG ∈ clique,
11: Opset (OPG) ∈ ListOfOpsets (bestFU) }
12:H1: size (maxCoveredOPGs) is maximum
13:H2: area (bestFU) is minimum
14: instantiate (bestFU); record(maxCoveredOPGs);
15:   foreach (clique ∈ listOfCliques)
16:   clique -=maxCoveredOPGs; --
```

C 10

Please replace the paragraph beginning at page 26, line 17 and ending at page 26, line 36 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

C 11

-- Many of these register file port connections may be shared based upon the mutual exclusion specification of the corresponding operation groups. As an example, assume that we want to build read/write port requirements for a machine specified by the following description:

```
SECTION Operation_Group {
OG_alu_0 (ops (ADD SUB) format(OF_intarith2));
```

```
        OG_alu_1 (ops (ADD SUB) format(OF_intarith2));


        OG_move_O (ops (MOVE) format (OF-intarithl));
        OG_move_1 (ops (MOVE) format (OF-intarithl));
    }
    SECTION Exclusion Group {
        EG_0 (opgroups (OG_alu_0 OG_move_0)
        EG_1 (opgroups (OG_alu_1 OG_move_1)
    }
    SECTION Register_File {
        gpr (width(32) regs (r0 rl ... r3l) virtual (I));
        pr (width(l) regs (p0 p1 ... pl5) virtual (P));
        lit (width(16) intrange (-32768 32767) virtual(L));
}--
```

C11

Please replace the paragraph beginning at page 27, line 1 and ending at page 27, line 11 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

C18

```
    -- SECTION Field_Type
        FT_I (regfile(gpr));
        FT_P) regfile(pr));
        FT_L (regfile(lit));
        FT_IL (compatible_with (FT_I FT_L));
}
```

I

```
SECTION Operation_Format {
    OF_intarithl (pred (FT P) src (FT_I) dest (FT_I));
    OF_intarith2 (pred (FT-P) src (FT_IL FT_I) dest (FT-I));
}--
```

Please replace the paragraph beginning at page 28, line 7 and ending at page 28, line 48 with the following paragraph that includes code that has a type size that is greater than .21 cm (.08inch) and line spacing greater than 1.5 lines:

```
-- Pseudo code for this resource allocation algorithm is listed below.
ResourceAlloc(nodeRequests, conflictgraph)
    // compute resource request for each node + neighbors
    foreach (node E conflictgraph) do
        Mark(node) = FALSE;
        TotalRequest(node) = Request(node) +
            Request(Neighbors0f(node));
            AllocatedRes(node) = empty
    endforeach
    //sort nodes by increasing remaining total resource request
    //compute upper-bound on resources needed by allocation
    resNeeded = 0; Stack = EMPTY;
    for (k from 0 to NumNodes(conflictGraph)) do

        find (minnode ∈ unmarked nodes) such that
```

```
            TotalRequest (minNode) is minimum;
            Mark(minNode) = TRUE;
            push(minNode, Stack);
            resneeded = max (resneeded, TotalRequest(minNode));
            foreach (nhbr ∈ NeighborsOf (minNode)) do
               TotalRequest(nhbr) -= Request(minNode);
            Endforeach
            endfor
        //process nodes in reverse order (i.e., decreasing total
        request)
        while (Stack is not EMPTY) do
            node = pop(Stack);
            AllResources = (O...resNeeded-1);
            // available resources are those not already allocated to
            any neighbor
            AvailableRes(node) = AllResources -
            AllocatedRes(Neighbors0f(node));
            //select requested number of port requests from available
            ports
            //according to one of several heruristics
            AllocatedRes(node) = Choose Request(node) resources from
            AvailableRes(node)
                H1: Contiguous Allocation
                H2: Affinity Allocation
```

C 13

```
end

return resneeded; --
```